Jesus L.C.

Recogiendo ideas

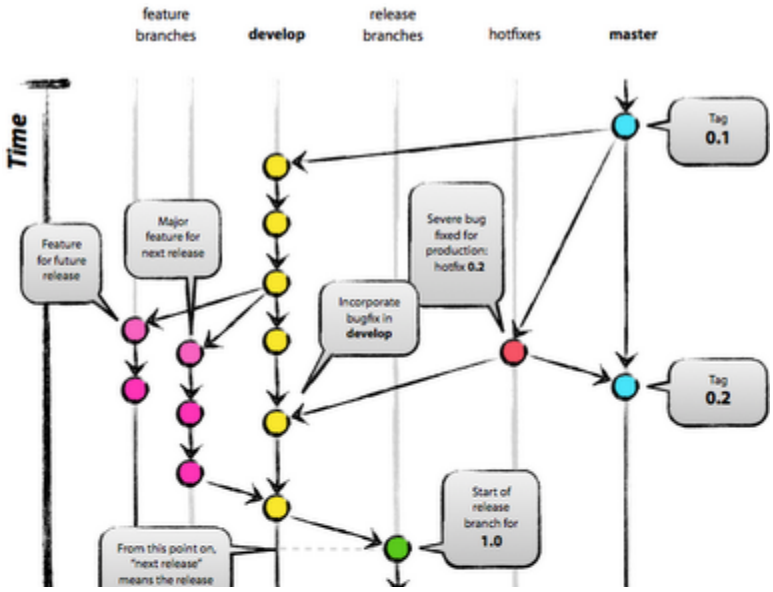
Una buena manera de afrontar la ramificación (branching) en Git

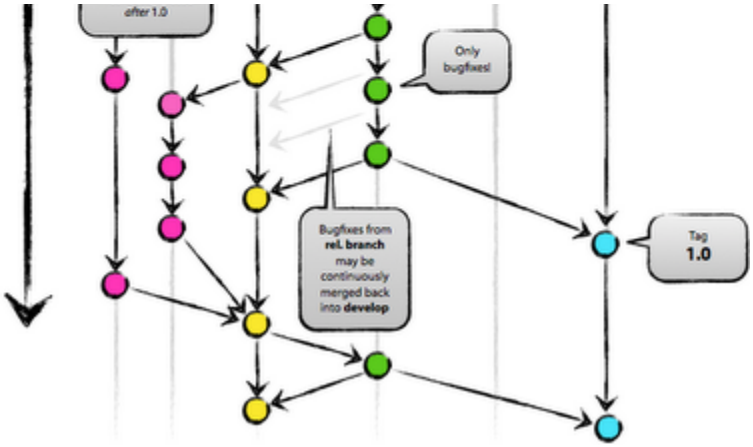
24 noviembre, 2012 por jesuslc

About these ads (http://en.wordpress.com/about-these-ads/)

Este post es una traducción de http://nvie.com/posts/a-successful-git-branching-model/ (http://nvie.com/posts/a-successful-git-branching-model/) en el que voy a contar una manera de trabajar con proyectos, Repositorios de código distribuidos (git) y ramas. Todo esto viene a que hace un tiempo encontré por barrapunto esta discusión (http://formacion.barrapunto.com/article.pl?sid=12/09/10/2248223) y encontré este post que ahora traduzco.

Estrategia de ramificiación y administración de versiones





¿por qué Git?

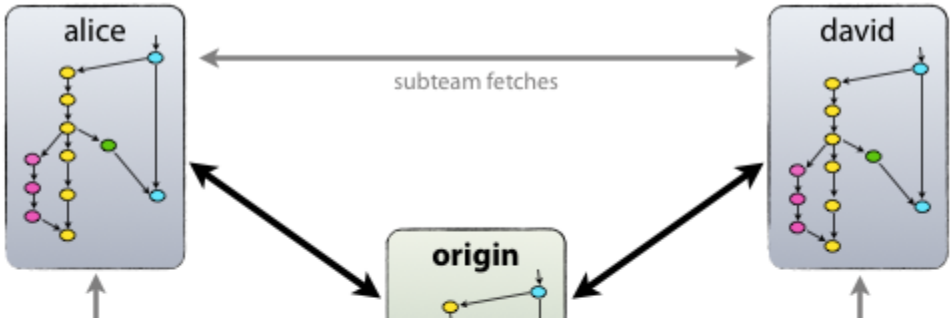
Existen en internet multitud de discusiones sobre los post y contras de Git, por ejemplo esta (https://git.wiki.kernel.org/index.php/GitSvnComparsion). Git da a los desarrolladores una nueva forma de pensar en fusión y ramificación. Desde CVS/Subversión siempre he tenido un poco de miedo por las bifurcaciones, sobre todo con los conflictos al mezclar (merge) *¡Los merges muerden!*

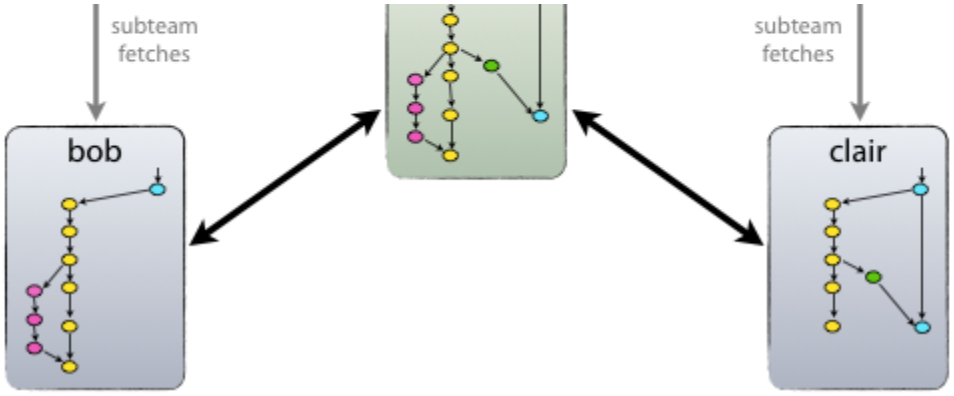
Pero con Git estas acciónes son muy naturales y sencillas. Por ejemplo si lees este libro (http://svnbook.red-bean.com/) sobre subversion el "branching" y "merging" se realiza en los últimos capítulos (Usuarios avanzados), mientras que en el libro de Git (http://git-scm.com/book/es) está en el capitulo 3.

Por tanto no debemos tener miedo por las ramas (branches) y las mezclas/fusiones (merges). Así que ahora vamos a hablar de un modelo de desarrollo, en realidad son una serie de procedimientos que deberían seguir los miembros de un equipo para tener un "proceso de software administrado"

Descentralizada pero centralizada

La configuración que utilizamos y que funciona viene con este modelo de ramificación, es tener un "repositorio central". Git es distribuido y no hay un repositorio central a nivel técnico, pero nos referimos a tener un repositorio **origin.**





Cada desarrollador realiza *pull* y *push* sobre el original. Pero ademas de los cambios en el "repositorio central" cada desarrollador también puede extraer cambios de otros compañeros y formar subequipos. Esto puede ser util para trabajar con con "miniequipos" o parejas de desarrolladores. (En la figura anterior hay 3 subequipos Alice y Bob, Alice y David, y Clair y David).

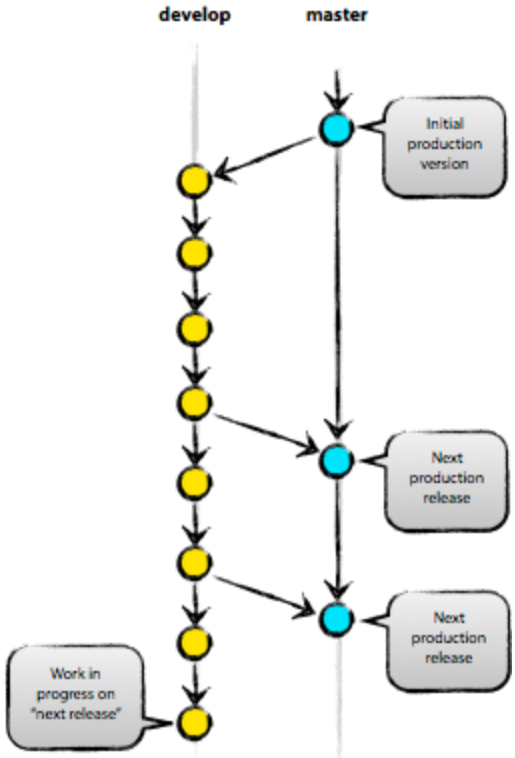
Técnicamente esto sólo quiere decir que Alice ha definido un Git remoto, llamado bob, señalando al repositorio de Bob y viceversa.

Las ramas principales

En el nucleo el modelo de desarrollo es clásico,

el repositorio central tiene 2 ramas principales:

- **master**
- **develop**



La rama ***master***debe ser un elemento familiar para todos los usuarios de Git. Paralelamente a la rama **master,** existe otra rama denominada **develop.**

Considerareéramos **origin/master** la rama principal, donde esta el código fuente de la **HEAD** siempre SIEMPRE siempre está listo para producción.

Consideramos **origin/develop** la rama principal, donde el código fuente de la **HEAD** siempre refleja un estado con los últimos cambios de entrega para el desarrollo de la próxima versión. (Hay gente que a esta rama la llama "rama de integración"). *En esta rama es donde se construyen las compilaciones automáticas.*

Cuando el código fuente que está en **develop** llega a un punto estable (totalmente probado y funcional) todos los cambios se incluyen en master y se

etiqueta **master** con un nuevo numero de versión(veremos más detalles de esto más adelante).

Por tanto cada vez que se combinan los cambios en **master** tenemos *por definición* una nueva versión en producción. En este paso debemos ser muy estrictos.

Ramas de apoyo

Junto a las ramas principales **master y develop** podemos tener una serie de ramas de apoyo para que nos ayuden en el desarrollo. A diferencia de lamas principales, estas tienen un tiempo de vida limitado.

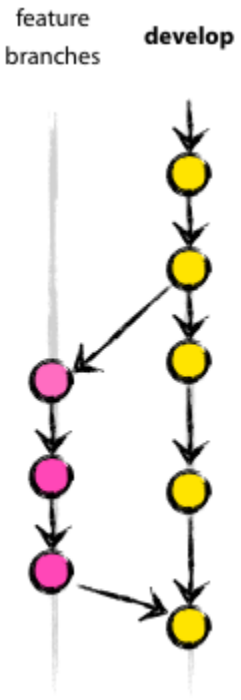
Los diferentes tipos de ramas que podemos utilizar son:

- Ramas de características *(features)*
- Ramas de lanzamiento *(Realise)*
- Ramas de revisiones (*hotfixes)*

Cada una de estas ramas tiene un propósito específico están sujetas a estrictas normas.

*Los tipos de ramas se clasifican por la forma en la que se utilizan.

Ramas de características



Estas ramas se utilizan para desarrollar nuevas características para una nueva versión. *La esencia de una rama de la característica es que existe, siempre y cuando la función está en desarrollo, pero finalmente se fusiona de nuevo en* ***develop*****.** Se ramifican a partir de **develop** y de combinan de nuevo en **develop**. Las ramas de características no existen en **origin.**

Creación de una rama de funcionalidad

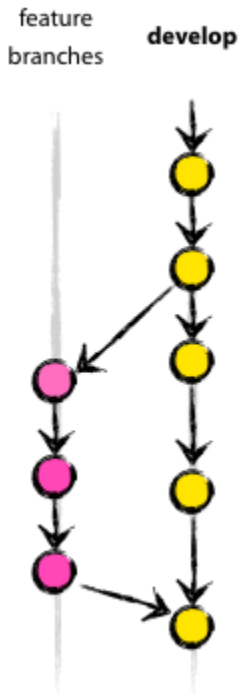
Esta es el comando ara crear una nueva funcionalidad, desde **develop**.

```
> git checkout -b myfeature develop
Switched to a new branch "myfeature"
```

Incorporación de una nueva característica en develop

```
> git checkout develop
Switched to branch 'develop'
> git merge --no-ff myfeature
Updating ea1b82a..05e9557 (Summary of changes)
> git branch -d myfeature
Deleted branch myfeature (was 05e9557).
> git push origin develop
```

La opción **–no** hace que se cree un nuevo objeto con cada commit, ara así no perder el histórico.



A ver si me explico, en este último caso es imposible saber cual de los objetos son los que tienen la nueva funcionalidad y en el caso de querer volver atrás habrá que leerse todo el registro de mensajes para deshacer la nueva característica. Por ello es más fácil utilizar la opción **–no-ff.**

*Hay que tener cuidado porque **–no-ff** no es el comportamiento predeterminado y las prisas pueden jugarnos una mala pasada.

Ramas de lanzamiento (release)

Se ramifican a partir de **develop.** Se deben combinar de nuevo en **develop** y **master.** La nomenclatura para estas branchs es **release-***

Las ramas de lanzamiento apoyan la preparación de una nueva versión para producción. Permiten "limar" los últimos detalles y corregir los errores menores Además de la preparación de los metadatos, fechas,…

Creación de una rama de lanzamiento

Las ramas de versión se crean a partir de **develop.** Por ejemplo, tenemos nuestro proyecto en la versión 1.1.5 y vamos a realizar un nuevo lanzamiento en breve. El estado de **develop** se convertirá en listo para el "próximo lanzamiento" u hemos decidido que la nueva versión será 1.2. Por tanto, se ramifica la rama de lanzamiento con un nombre que refleje el número de versión:

```
> git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"
> ./bump-version.sh 1.2
Files modified successfully, version bumped to 1.2.
> git commit -a -m "Bumped version number to 1.2"
[release-1.2 74d9424] Bumped version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)
```

Después de crear una nueva rama y cambiar a ella, nos topamos con el número de versión. Aquí **bump-version.sh b** es un script de trabajo que cambia algunos archivos de la copia de trabajo para reflejar la nueva versión (esto puede ser un cambio manual lo importante es que *algunos* archivos cambien) Entonces, el numero de versión cambia.

Esta nueva rama va a existir durante un tiempo, hasta que se libere la nueva release. Durante ese tiempo la corrección de errores DEBE aplicarse en esta rama en vez de en **develop.**

*No se deben añadir grandes caracteristicas en esta rama, es necesario esperar a otro lanzamiento.

Terminar un lanzamiento

Cuando la rama de lanzamiento está lista para llevarse a producción (liberarse) es necesario realizar una serie de acciones. En primer lugar, la nueva rama se fusiona con **master** (ya que por definición cada commit en **master** es una nueva versión). A continuación, se deben tagear/documentar los nuevos cambios para tener referenciado el histórico de esta versión. Por último, los cambios realizados en la rama de lanzamiento deben ser incluidos de nuevo en **develop**, con el fin de que futuras versiones también contengan estas correcciones de errores.

Los dos primeros pasos de Git son:

```
> git checkout master
Switched to branch 'master'
> git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
> git tag -a 1.2
```

El lanzamiento y los etiquetados se realizan ahora. Para mantener los cambios en la rama de lanzamiento, tenemos que unir lo de **develop**

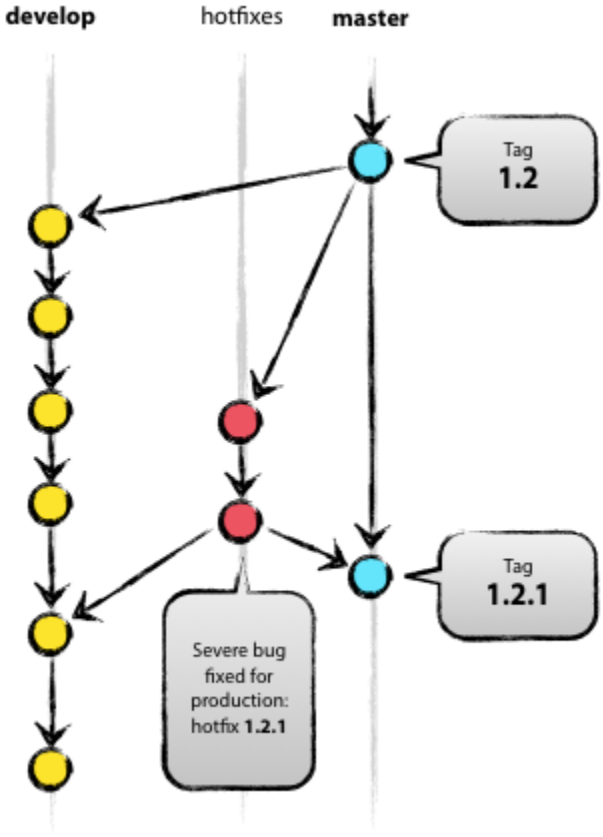
```
> git checkout develop
Switched to branch 'develop'
> git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
```

Este paso puede dar algún conflicto de combinación, si es así debemos fix and commit.

Ahora ya estamos llegando al final y la rama de release debe ser eliminada, ya que no la vamos a necesitar más.

```
 >git branch -d release-1.2
Deleted branch release-1.2 (was ff452fe).
```

Ramas de revisiones (hotfixes)



Se ramifican a partir de **master** y se deben combinar de nuevo en **develop** y **master.** La nomenclatura debe ser **hotfix.**

Las ramas de revisión son muy similares a las ramas de release, ya que ambas están destinadas para producción, aunque estas (htfixes) no están planificadas.

Si un bug es crítico debe ser resuelto inmediatamente. La esencia de este me método de trabajo es que los miembros del equipo (rama **develop**) pueden seguir trabajandomientras que otra persona soluciona el bug.

Creación de una rama hotfix

Las ramas de revisión se crean a partir de la rama **master.** Veamoslo con un ejemplo, digamos que estamos en la versión de producción 1.2 y esta hay un bug bastante grave. supongamos que los cambios en **develop** son todavía bastante inestable. A continuación vamos a bifurcar la rama **master** para arreglar el problema.

```
> git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
> ./bump-version.sh 1.2.1
Files modified successfully, version bumped to 1.2.1.
```

```
> git commit -a -m "Bumped version number to 1.2.1"
[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1
1 files changed, 1 insertions(+), 1 deletions(-)
```

No hay que olvidarse de modificar el número de versión despueé de la ramificación.

Después, arreglar el fallo y confirmar la corrección con uno o mas commits separados.

```
> git commit -m "Fixed severe production problem"
[hotfix-1.2.1 abbe5d6] Fixed severe production problem
5 files changed, 32 insertions(+), 17 deletions(-)
```

Terminar una rama de hotfix

Cuando se termine la corrección de errores, las soluciones deben ser incluidas de nuevo en la rama **master,** pero tambien deben incluirse en la rama **develop** con el fin de salvaguardar todos los cambios. Este paso es similar la liberación de una rama despues de una release.

En primer lugar actualizamos **master** para marcar la liberación.

```
>git checkout master
Switched to branch 'master'
> git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
> git tag -a 1.2.1
```

Después incluimos este “parche” también en **develop**

```
> git checkout develop
Switched to branch 'develop'
> git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)
```

La única excepción de la regla es que, cuando existe una rama release, los cambios en las revisiones se fusionaran en esa rama de lanzamiento en lugar de **develop**.

Por ultimo matamos la rama temporal

```
> git branch -d hotfix-1.2.1
Deleted branch hotfix-1.2.1 (was abbe5d6).
```

Resumen

Aunque no hay nada nuevo, impactante y/o asombroso en este modelo de ramificación, tener este modelo puede ser realmente útil en nuestros proyectos. Con este texto es fácil construir un modelo mental que permite a los miembros del equipo entender los procesos de ramificación y liberación.

Escrito en desarrollo, software | Etiquetado branch, fusion, git, hotfix, master, merge, rama por tarea, ramas, release | 1 comentario

Una respuesta

1. *en 25 noviembre, 2012 a 10:42 pm* | *Responder*  *Jorge Jiménez (@jorgejiro)*
   Te aconsejo este enlace, que facilita mucho la utilización de gitflow:
   https://github.com/nvie/gitflow

RSS de los Comentarios

Seguir

Follow “Jesus L.C.”